Procedia Computer Science

Volume 29, 2014, Pages 509–522

ICCS 2014. 14th International Conference on Computational Science

A comparative study of scheduling algorithms for the multiple deadline-constrained workflows in heterogeneous computing systems with time windows

Klavdiya Bochenina[1]

[1] *ITMO University, St. Petersburg, Russia*

k.bochenina@gmail.com

Abstract

Scheduling tasks with precedence constraints on a set of resources with different performances is a well-known NP-complete problem, and a number of effective heuristics has been proposed to solve it. If the start time and the deadline for each specific workflow are known (for example, if a workflow starts execution according to periodic data coming from the sensors, and its execution should be completed before data acquisition), the problem of multiple deadline-constrained workflows scheduling arises. Taking into account that resource providers can give only restricted access to their computational capabilities, we consider the case when resources are partially available for workflow execution. To address the problem described above, we study the scheduling of deadline-constrained scientific workflows in non-dedicated heterogeneous environment. In this paper, we introduce three scheduling algorithms for mapping the tasks of multiple workflows with different deadlines on the static set of resources with previously known free time windows. Simulation experiments show that scheduling strategies based on a proposed staged scheme give better results than merge-based approach considering all workflows at once.

Keywords: **deadline-constrained workflows, workflow scheduling, time windows**

1 Introduction

Over the past decade, the problem of scheduling interrelated tasks in heterogeneous distributed systems (such as grids and clouds) has gained particular attention due to the increased use of scientific workflows in a variety of subject areas in conditions of continuous growth of available computing power. Today there are a lot of static and dynamic scheduling approaches designed for different combinations of workflows and resources properties (e.g., parallelism level, graph shape, data interchange intensity, arrival patterns of tasks, homogeneity of processors' performance and communication speed, on-demand resources accessibility). Goals of scheduling process are

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.046

determined based on both the selected infrastructure architectural concepts [1] (possibilities given by resource providers) and customers' requirements (QoS limitations).

In general, modern workflow management systems operate under conditions of high unpredictability both of workload and resource accessibility, which leads to the necessity to use dynamic scheduling algorithms for virtualized environments. However, for problems in which we have good estimates of tasks' execution times, exactly know the information about resource set compound, available times of resources and moments when workflows start, it is advisable to use static scheduling methods which can provide better schedules than dynamic ones due to taking into account a workflow structure. The shining example of such a problem is a regular operating process of an urgent computing system when periodically incoming pieces of data should be processed using workflow mechanism without violating user defined deadline [2] (which can coincide with the moment of receiving next chunk of data). For instance, this scheme is commonly used in urban flood decision support systems [3], [4]. Typical modeling workflow consists of 5-25 tasks and can include stages of meteorological data processing, simulation and prediction of sea level and wave parameters, and various decision support scenarios. New instance of regular operating process is started every several hours to obtain up-to-date forecast based on a recent data. Non-violating the deadline of this process guarantees a relevance of the results which is crucial for an effective flood prevention.

In normal operation mode the regular processes often use a specific static set of resources which can be non-dedicated to workflows execution. At the same time, information about available CPU time windows can be provided in advance, i.e. if one reserves computational capacities for scientific or educational purposes. Also, periodical nature of regular calculating process provides an opportunity to gather a comprehensive statistics about execution times of single tasks. In case of several processes (each with its own data incoming frequency and its own maximum makespan) the scheduling problem is to find a mapping of tasks to processors maximizing the efficiency of resource utilization while meeting all workflows deadlines (and maybe some other users' constraints).

The contributions of this paper are as follows: 1) formulation of the scheduling problem (shortly described above); 2) development of three algorithms for scheduling of multiple deadline-constrained workflows in heterogeneous platforms with time windows; 3) simulation experiments on the sets of synthetic workflows to compare the effectiveness of these algorithms according to the proposed metrics.

2 Related works

Over the past decade, there have been proposed a lot of algorithms in the area of single deadline-constrained workflow scheduling. Most of these algorithms are designed to solve the problem of minimizing the workflow execution cost/makespan while respecting their service level agreements (SLAs) to ensure the QoS compliance, and moreover they are focused on dynamic resource allocation using virtual machines. Bossche et al.[5] and Genez et al. [6] described the formulation of this problem in terms of integer linear programming and proposed a set of heuristics to solve it. Similar problem statement can be found in [7], where authors introduced an SCB (Server Count Bound) heuristic to find the minimum resource count which is required to execute workflow without violating the deadline. Yu et al. [8] studied single workflow scheduling with time and cost constraints and proposed a genetic algorithm based on heuristic for minimizing execution time while meeting user's budget constraints. Zheng and Sakellariou [9] proposed a heuristic BHEFT as an extension of HEFT algorithm for market-oriented environments with non-dedicated resources. In the paper [10] authors introduced a novel single workflow scheduling algorithm PEFT which outperforms well-known list-based heuristics in terms of makespan, efficiency and frequency of best results.

Many researchers use the idea of assigning sub-deadlines to individual tasks (if we meet all the sub-deadlines, we will automatically meet the whole workflow deadline). Yu et al. [11] and Yuan et

al. [12] suggested to group the tasks according to their level (i.e., path length from the input task) and to find a schedule meeting group deadlines (Deadline Top Level, DTL and Deadline Bottom Level, DBL algorithms). Scheduling method proposed in [13] is also based on tasks grouping. Wang et al. [14] developed a heuristic algorithm Deadline Early Tree (DET), where tasks of critical path are scheduled using dynamic programming. Determination of the certain tasks deadlines is usually produced using one of two approaches: 1) providing the earliest starting time and latest finish time using the network planning methods, 2) setting the deadlines proportional to the computational complexity of tasks. Thus, the Partial Critical Path (PCP) algorithm [15] initially finds the deadlines for the critical path tasks, and then uses a recursive procedure to set other tasks' deadlines. Later two modifications, IC-PCP and IC-PCPD2, has been developed in [16].

Only few studies have considered multiple workflows scheduling problem. As showed in [17], there are two main approaches to schedule a set of workflows: 1) merging the tasks of all workflows into a single composite workflow and applying any single workflow scheduling algorithm (e.g., SOT (Serve on Time algorithm)[18]); 2) ordering single workflows according to some metric and then scheduling it consequently (e.g., FPFT/FPCT (Fairness Policy based on Finishing/Concurrent Time) algorithms [19]). Mao et al. [20] proposed an algorithm to minimize the multiple workflows' execution cost while meeting the deadlines, intended for scalable virtualized computational environment. In [21] there is proposed an approach to cost- and deadline-constrained scheduling of ensembles of inter-related workflows in 'resources-on-demand' cloud environment.

For scientific workflow scheduling on partially available resources with preliminary reserved time windows, Luo et al. [22] proposed three algorithms for single data-parallel workflows (exact branch-and-cut based and two heuristics, MHEFT-RSV and MHEFT-RSV-BD).

To the author's best knowledge, development of multiple deadline-constrained workflows' scheduling algorithm on heterogeneous platforms with predefined time windows is still an open research problem, and no conclusive results have been reported in the literature on this topic.

3 Multiple deadline-constrained workflows scheduling with time windows: problem formulation

Scientific workflows typically use two types of parallelism which can be exploited to get a higher benefit from the large computing power [23]: *task parallelism* (when different tasks of a particular workflow can be executed concurrently) and *data parallelism* (when each task can be executed on more than one processor at the time). Task-parallel workflows are usually represented as DAG (Directed Acyclic Graph) while mixed (task and data parallel) applications use PTG (Parallel Task Graph) model. In this paper we consider DAG representation of workflows with non-preemptive tasks, so one task can be executed only on one processor without suspending and migrating to other processor. Thus, each workflow in the set $WFSet$ can be represented as a graph $WF_i = (V_i, E_i), i = \overline{1, N_{WF}}$ where N_{WF} – number of workflows in $WFSet$, $V_i = \{1,...,Np_i\}$ – indexes of tasks of i-th workflow, Nt_i – number of tasks of i-th workflow, $E_i = \{e_{ij} \mid i, j \in V_i \times V_i\}$ – a set of edges representing links between workflow tasks. $WFSet$ is a set of tasks $Task_{ij}, i = \overline{1, N_{WF}}, j = \overline{1, Nt_i}$. Each task can be executed on some set of resource types $RT_{ijk}, k = \overline{1, Nrt_{ij}}$ (Nrt_{ij} is number of resource types for j-th task of i-th workflow). Also, estimates of execution times t_{ijk} are known for each possible combination of task and resource type. This estimates can be obtained by preliminary profiling, handling data from previous runs or using parametric performance models of particular tasks

[24]. If performances of particular resources (in FLOPS) and FLOP count for each task are specified, we can use this information to calculate approximate estimates.

Given a computer platform that consists of N_R resources related to different N_T resource types (where $NR_i, i = \overline{1, N_T}$ – number of resources of i-th resource type). Different resources are assigned to one resource type, if they have the same technical characteristics (processors/cores count, RAM size) and there is a high-speed network connection between them. In this work we assume that given workflows are not communication-intensive, so we can neglect a data transfer time.

Let T be the length of planning period. All workflows have the known start times $tbegin_i, 0 \le tbegin_i \le T$ and deadlines $deadline_i, tbegin_i \le deadline_i \le T$ where $i = \overline{1, N_{WF}}$. The simplest way to measure the efficiency of resources utilization is to find a ratio of time occupied by scheduled tasks to initially available time. In this case, time intervals with the same length have the same value of objective function regardless of these placements. We consider more sophisticated case when effectiveness of resource usage is characterized by *utility function* $g(\tau)$ which reflects preferred loading time ($\tau = \frac{t}{T} \in [0;1]$ – normalized time). If we assume that the maximum resource utilization level is equal to 1, and all resources have the same utility function, then

$$\int_0^1 g(\tau)d\tau = \frac{1}{\sum_{i=1}^{N_R} Nproc_i}, \tag{3.1}$$

where $Nproc_i$ – number of processors for i-th resource type. Total efficiency of resource utilization for resource type R_i in the period $[\tau_1;\tau_2]$ is

$$W(R_i, \tau_1, \tau_2) = \sum_{j=1}^{Nproc_i} \sum_{k=1}^{Int_{ij}} \int_{\tau_{ijk}^b}^{\tau_{ijk}^e} g(\tau)d\tau, \tag{3.2}$$

where $Int_{ij} = Int_{ij}(\tau_1, \tau_2)$ – number of intervals occupied by scheduled tasks for j-th processor of i-th resource type in the period $[\tau_1;\tau_2]$, $\tau_{ijk}^b, \tau_{ijk}^e$ – beginning and end of the k-th interval.

In this work we use a simple decreasing linear function as utility function:

$$g(\tau) = \frac{2}{\sum_{i=1}^{R_i} Nproc_i} \cdot (1 - \tau). \tag{3.3}$$

So, for the same resource type, the earlier is the start time of a particular task, the greater is the corresponding resource utilization.

The goal of scheduling is to maximize the overall resource utilization level for a given period $[0;T]$ without violating deadlines of workflows ($tend_i$ is a finishing time for i-th workflow in the obtained schedule):

$$W = \sum_{i=1}^{Nr} W(R_i, 0, 1) \to \max, \tag{3.4}$$

$$\forall Wf_i, i = \overline{1, N_{WF}} \quad tend_i \le deadline_i. \tag{3.5}$$

4 Considered scheduling algorithms

As we mentioned before, there are two main approaches to multiple workflows scheduling (the one is based on merging all tasks into a big workflow, and the other takes into account affiliation of tasks to workflows). Our hypothesis is that in the case of deadline-constrained workflows the second approach should give better results since it becomes possible to consider the particular deadlines. To check it, we have developed algorithms implementing merge-based and workflow-based approaches. Note that they both should use some additional algorithm for a single workflow scheduling.

4.1 Single deadline-constrained workflow scheduling algorithm for heterogeneous resources with time windows

Since violation of the deadline is undesirable, the purpose of an algorithm is to maximize the time between actual workflow completion and its deadline (on the other hand, increasing this margin allows to increase the possibility of meeting the deadline if some tasks finish their execution later than planned). More formally, the scheduling goal for a given Wf_i, $i = \overline{1, N_{WF}}$ is to maximize $deadline_i - \max_j \left(tend\left(P_{ij}\right) \right)$ where $tend\left(P_{ij}\right)$ is the finishing time of j-th task. There are a lot of well-known and efficient algorithms for minimizing the makespan of workflow execution on a set of heterogeneous resources (such as HEFT and PEFT[10]). In this work we propose list-based heuristic *SimpleSched* which is similar to those algorithms due to basic steps (prioritizing and assigning tasks to workers). The main difference is that we use the value of deadline while building a queue of tasks. It is crucial to consider the deadlines of particular workflows in the algorithm because our merge-based approach is based on scheduling tasks from different workflows at once, and priority of the task should reflect accumulated execution time (or computational cost) as well as required finishing time of the workflow.

Then, *SimpleSched* heuristic consists of the following steps:

1) Determination of a tentative latest finishing time (sub-deadline) for each task;

2) Creation of a priority queue of tasks considering sub-deadlines and tasks' precedence constraints;

3) Mapping each task from queue to the resource which provides the earliest finishing time for its execution.

If estimation times t_{ijk} are given, evaluation of latest finishing times can be done by network analysis methods (note that if the minimum earliest finishing time is greater than the given deadline, workflow cannot be finished in time). Consider the case when we have information about resources performance and FLOP count for each task.

Let $LFT(i)$ be the latest finishing time of i-th task of a given workflow, $calc(i)$ – computational cost of i-th task (in FLOP), $amount(i)$ – maximum accumulated computational cost from initial task to i-th task (including its own cost), $in(i)$ and $out(i)$ – sets of input/output indexes of tasks for i-th task. Then defining the sub-deadlines for all tasks can be described as follows ($used$ is a set of task indexes with already calculated values of $amount$, $linked$ is a list of tasks for which at least one of its input tasks is in $used$ set, $linked \cap used = \varnothing$). The description of the algorithm for calculating the sub-deadlines for all tasks of a single workflow is given in Table 1.

1. $used = \{i : in(i) = \varnothing\}$, $linked = \{j : i \in used, j \in out(i)\}$.
2. $\forall i \in used \quad amount(i) = calc(i)$.
3. While $linked \neq \varnothing$ repeat steps 4-6.
4. $current = linked_0$.

5. $linked \leftarrow linked \setminus current$.
6. If $in(current) \subseteq used$, $amount(i) = \max_{j \in in(current)} (amount(j)) + calc(i)$, $used \leftarrow used \cup current$, $\forall i \in out(current)$, $i \notin linked$ $linked \leftarrow linked \cup i$. Else $linked \leftarrow linked \cup current$.
7. Find i^* such that $\forall j \in used$ $amount(i^*) \geq amount(j)$.
8. $LFT(i^*) = deadline$.
9. $\forall i \in used, i \neq i^*$ $LFT(i) = \frac{amount(i)}{amount(i^*)} \cdot deadline$.

Table 1. Algorithm for calculating the sub-deadlines of workflow tasks by their computational costs

When the latest finishing times are calculated, tasks are placed in priority queue in ascending order of its sub-deadlines (it is also necessary to take into account that package should not be scheduled before any of its input packages). After construction the queue packages are mapped consequently.

Let $EST(Task_{ij}) = \max\left[tend(pred(Task_{ij}))\right]$ be the earliest possible start time of i -th task of the j -th workflow as a maximum finishing time of its already scheduled predecessors, and $EST(Task_{ij}, R_k) \geq EST(Task_{ij})$ – the earliest start time of this task on the k -th resource type considering actual busy time windows. The task assigns to the resource R_{k*} which provides earlier finishing time for it ($R_{k*} : EFT(Task_{ij}, R_{k*}) = \min_k \left\lfloor EST(Task_{ij}, R_k) + t_{ijk} \right\rfloor$).

In fact, the merge-based algorithm is *SimpleSched* heuristic (described above) applied to workflow constructed from all available packages. Since packages belonging to different workflows are not linked, one can calculate sub-deadlines independently for each particular workflow, and then merge all packages into queue. So, merge-based algorithm implicitly takes into account workflow deadlines by primarily scheduling tasks with earlier sub-deadlines, but in a situation when several tasks of different workflows have similar LFTs and compete for resources (due to their lack) it provides no guarantees that resources will be given to packages which workflow has earlier deadline.

To illustrate previously mentioned statement about necessity of taking into account the deadlines while prioritizing the tasks of multiple workflows at once, we consider the example listed below.

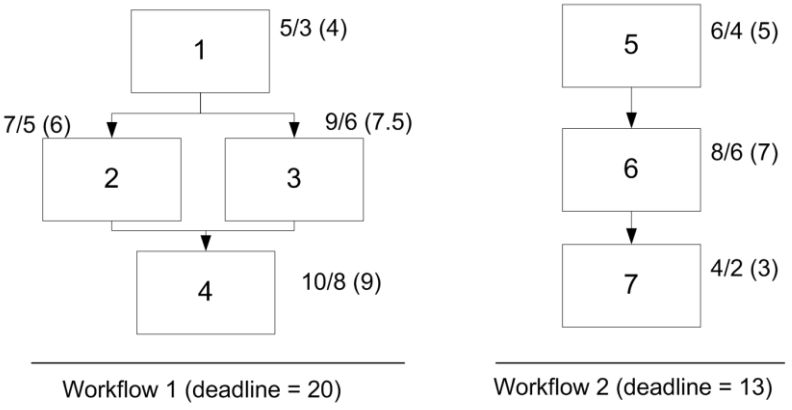

Figure 1. Test example (2 workflows). Notation: a/b (c),
a - time of execution on R1, b - time of execution on R2, c - average execution time

Assume that we have two workflows as shown at Figure 1, and a set of 3 resources (1 resource of type 1 (R1), and 2 resources of type 2 (R2)). Let's compare the results of HEFT (without taking into account the communication costs) and SimpleSched algorithms.

Table 1 shows the ranks of tasks according to HEFT algorithm and LFTs of tasks according to SimpleSched. The ranks for HEFT are calculated using the formula:

$$rank(Task_i) = \overline{w_i} + \max_{j \in succ(Task_i)} rank(Task_j), \quad (4.1)$$

where $\overline{w_i} = \frac{a_i + b_i}{2}$ – an average execution time of i-th task.

Task index	Rank (HEFT)	LFT (SimpleSched)
Workflow 1		
1	20.5	3.9
2	15	9.7
3	16.5	11.2
4	9	20
Workflow 2		
5	15	4.3
6	10	10.4
7	3	13

Table 1. Values of prioritization criteria for HEFT and SimpleSched

Due to Table 1, the order of assigning the tasks to processors will be: i) for HEFT: 1, 3, 5, 2, 6, 4, 7; ii) for SimpleSched – 1, 5, 2, 6, 3, 7, 4.

The results of assigning the tasks to processors with earliest finishing time for each task are summarized on Figure 2.

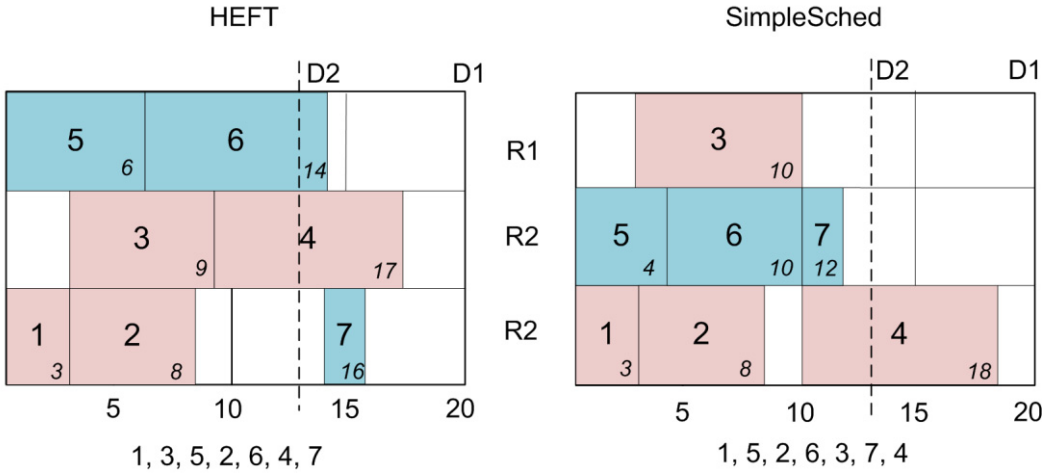

Figure 2. SimpleSched vs HEFT for the sets of deadline-constrained workflows

For the test example HEFT provides the schedule where Task6 and Task7 violate the deadline, and SimpleSched gives the schedule without violating the deadline. At the same time, makespan for workflow 1 increases in case of SimpleSched only by 1 time unit while workflow 2 reduces its makespan by 4 time units in comparison with HEFT.

If we suppose that overall number of workflows is n, and each workflow contains m tasks, the computational complexity of *SimpleSched* will be $O(m \cdot n)$.

4.2 Stage-based approach to a multiple deadline-constrained workflow scheduling with time windows

The main idea of the second approach is to consider workflow deadlines as much as possible. If one is going to schedule multiple workflows consequently, one should know the order in which workflows will take access to resources selection (in other words, it is necessary *to prioritize* workflows). The most obvious solution of the problem is to assign priorities according to its deadlines. This approach does not seem to be the best for several reasons including possible presence of multiple workflows with the same or similar deadlines which have considerably different computational complexities and/or starting times.

Suppose that we are given *prioritization criteria*, i.e. a certain metric applicable to workflow schedule which extreme value is used to choose a workflow for resource allocation. The main concept of stage-based approach is that scheduling procedure consists of several steps (by number of workflows), and each step involves a choice of one workflow in accordance with prioritization criteria, scheduling it and fixing busy time windows on selected resources. In ordered scheme, a particular workflow can be scheduled with any algorithm for minimizing makespan of a single workflow (i.e., HEFT). There is only one restriction that selected algorithm should assign the tasks to the processors with consideration of reserved time windows.

Let *Scheduled* be the set of workflows' numbers that were already scheduled, *Unscheduled* – the set of workflows' numbers that require scheduling, *Intervals* – current set of resources' busy intervals, $Schedule_i(Intervals)$ – schedule for i-th workflow taken by single workflow scheduling algorithm for current time windows, $Priority(Schedule_i)$ – value of prioritization criteria of $Schedule_i$. Assume that workflow selection is carried out according to minimum prioritization criteria value. Then an algorithm implementing stage-based approach (it is denoted as *StagedScheme*) can be described as follows.

1. $Scheduled = \varnothing$, $Unscheduled = \varnothing$, $Intervals \leftarrow Intervals_{init}$, where $Intervals_{init}$ is initial busy time windows.
2. For all $Wf_i, i = \overline{1, N_{WF}}$ $Unscheduled \leftarrow Unscheduled \cup i$.
3. For all $i \in Unscheduled$ get the schedule $Schedule_i(Intervals)$.
4. Find $i^* = \{i : \forall i \neq j \; Priority(Schedule_i) \leq Priority(Schedule_j)\}$ where $i, j \in Unscheduled$.
5. $Scheduled \leftarrow Scheduled \cup i^*$, $Unscheduled \leftarrow Unscheduled \setminus i^*$.
6. $Intervals \leftarrow Intervals \cup Schedule_{i^*}$.
7. Repeat steps 3-6 while $Unscheduled \neq \varnothing$.

Table 2. The StagedScheme algorithm for a multiple deadline-constrained workflows scheduling with time windows

If we suppose that computational complexity of scheduling one workflow of m packages is $O(m)$, and we should schedule n workflows of m packages, the computational complexity of *StagedScheme* is $O(m) \cdot n + O(m) \cdot (n-1) + \ldots + O(m) \cdot 1 = O(m) \cdot \sum_{i=1}^{n} i = O(m) \cdot \frac{n \cdot (n+1)}{2} \sim O(m \cdot n^2)$.

5 Simulation technique

To study the effectiveness of proposed scheduling algorithms we have generated synthetic test data which parameters and values are summarized in Table 3. The goal of the simulation was to compare the merge-based and ordered schemes under the circumstances of wide variance of number of workflows and tasks on the static set of resources. This approach can show us the main tendencies in behavior of studied algorithms by fixing the parameters of resources and varying the amount of computations and the partition of these computations into workflows and tasks.

Workflows' test examples were obtained with DAGGEN workflow generator which allows us to generate random graphs of tasks with different shapes of graphs. The most important parameters affecting the shape of the graph are width (maximum number of tasks that can be executed concurrently), density (number of dependencies between tasks of consecutive graph levels), regularity (regularity of the distribution of tasks between the different levels) and jump (maximum number of levels spanned by task communications). Obtained test examples were supplemented with the values of package computational cost which were chosen so that on a particular set of resources one package has execution time in the range of 300-43200 sec. To receive test examples of resources we implemented a resource generator which provides sets of non-dedicated resources with heterogeneous processors performances. We' ve chosen so high level of heterogeneity of resources (ten times difference as a maximum) because modern systems often include not only CPUs but GPUs as computing devices, and inequality of performance parameters of these devices can be significant.

Parameter name	**Possible values**
Workflows parameters	
Workflows count	25; 50; 75; …; 400
Task computational cost (GFLOP)	15000-216000
Number of tasks per workflow	5; 20; 50
Width of the DAG	0.1; 0.2; 0.8
DAG density	0.2; 0.8
DAG regularity	0.2; 0.8
DAG jump	1, 2, 4
Resources parameters	
Processor performance (GFLOPS)	5-50
Part of non-dedicated time per processor	0.25; 0.5; 0.75
Non-dedicated intervals count per processor	0-3
Number of resources types	1; 2; 4; 8
Number of resources per resource type	1; 2; 4; 8; 16
Number of processor per resource	1; 2; 4

Table 3. Parameters of Synthetic Test Data

Estimation of comparative efficiency of algorithms was made by the calculation of several metrics:

1. Average reserved time

$$R_{time} = \frac{\sum_{i=1}^{N_{WF}} reserve\_time_i}{N_{WF}}, \quad (5.1)$$

where $reserve\_time_i = \begin{cases} 0, deadline_i \le tend_i \\ deadline_i - tend_i, deadline_i > tend_i \end{cases}$.

2. Violated deadlines per number of workflows in the set.

3 Loading efficiency

$$Eff = \frac{W\left(Intervals_{busy}\right)}{W\left(Intervals_{free}\right)}, \tag{5.2}$$

where $Intervals_{busy}$ – time windows used by workflows, $Intervals_{free}$ – initial free time windows.

4. The execution time of algorithm.

Each set of experiments was carried out for a specific number of tasks per workflow (for example, 20) and different number of workflows (from 25 to 400). Each workflow set included approximately the same number of workflows for different DAG widths. Deadlines of all workflows were equal to the end of planning period. All experiments were performed on the same set of resources (8 resource types, 16 resources per type, 4 processors per resource, part of non-dedicated time per processor – 0.25) to get the most precise estimates of execution times of the algorithms.

Each experiment was carried out twice for two different planning periods. First length of the period (denoted as 'wide' makespan) has been selected so that all workflows in maximum set (i. e. 400 workflows) could finish execution without violating the deadlines. To limit a wide makespan, we suppose that loading efficiency should be at least 0.8 for the maximum workflows' set. Experiments with wide makespan were used to estimate average reserved times and loading efficiencies. Second planning length of the period (denoted as 'tight' makespan) was taken as 75% from wide makespan. It has been used to estimate the part of violated deadlines per workflow count.

6 Simulation results

We denote the studied algorithms as follows: 1) *Merge-based* – an algorithm implementing merge-based approach; 2) *MaxReserved* – an algorithm implementing the staged scheme and using maximum reserved time as a prioritization criteria; 3) *MinEff* – an algorithm implementing the staged scheme and using minimum efficiency as a prioritization criteria. All three algorithms use *SimpleSched* for a single workflow scheduling. Since a single set of experiments was carried out for a particular number of tasks per workflow, we use this value to denote the set (for example, *MaxReserved 20*).

In Figure 3 we show a ratio of the average reserved time (which was calculated using (5.1)) to the makespan. Note that in all cases *MinEff* provides better reserved times than two other algorithms, especially for the set of 5-tasks workflows where its maximum advantage is up to 13%. *Merge-based* algorithm shows the lowest values and faster decrease of reserved time than stage-based algorithms in all sets of experiments.

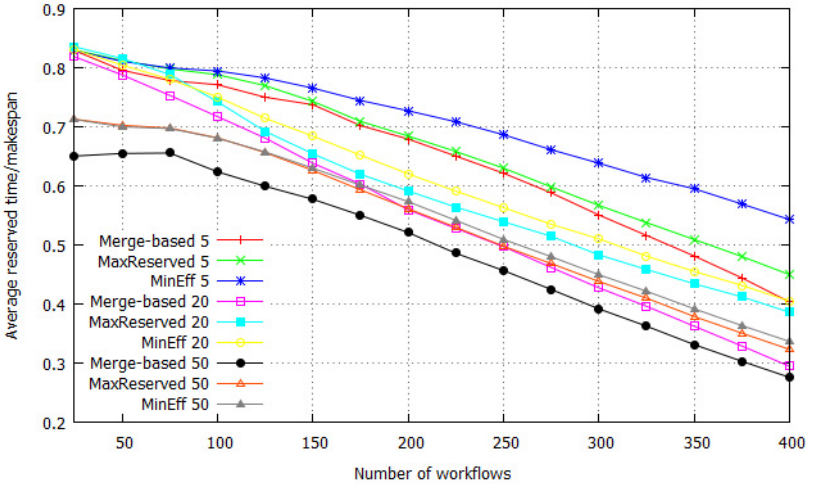

Figure 3. Ratio of the average reserved time to makespan

Figure 2 shows the efficiencies taken for a wide makespan. Staged-based algorithms demonstrate near efficiencies for all experiments (except MinEff 5). Maximum advantages for MaxReserved algorithm in comparison with Merge-based is respectively 6,79% for 5-tasks sets, 6.36% for 20- tasks sets and 14.91% for 50- tasks sets. Note that on both figures described above maximum differences between algorithms appear for experiments with number of workflows more than 200. It can be explained by the fact that the increase of number of workflows (with the same resource set) leads to the increase of competition for resources.

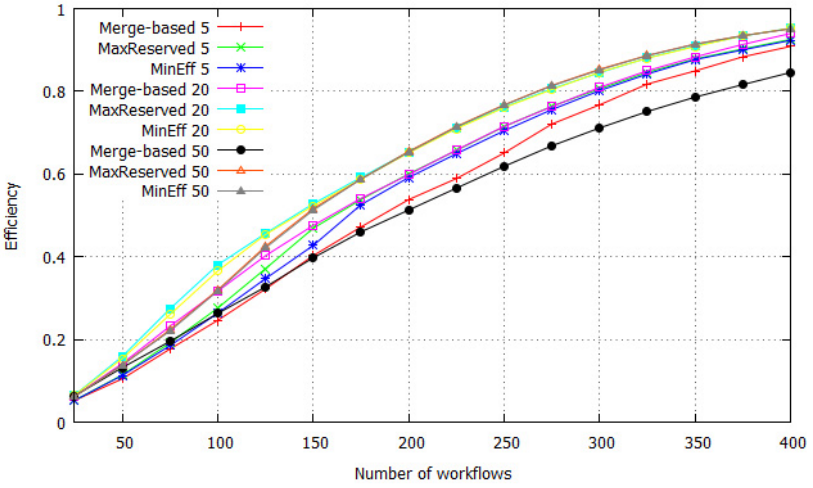

Figure 4. Efficiencies for wide makespan

Figure 5 shows the ratio of violated deadlines to workflows count for a 50- tasks test set for a tight makespan. *Merge-based* algorithm demonstrates absolutely the worst results with maximum disadvantage equal to 32,8% and average disadvantages equal to 19,28% for *MaxReserved* and 15,2% for *MinEff*. Efficiency estimates have shown that loading for 275 workflows (and higher) was more than 95% for staged-based algorithms. It explains a sharp rise of the metric beginning at that point (there were no more available time slots). We should note that loading efficiency of *Merge-Based* was in average 10% less than of both *MaxReserved* and *MinEff* (despite the fact that it had more unscheduled tasks).

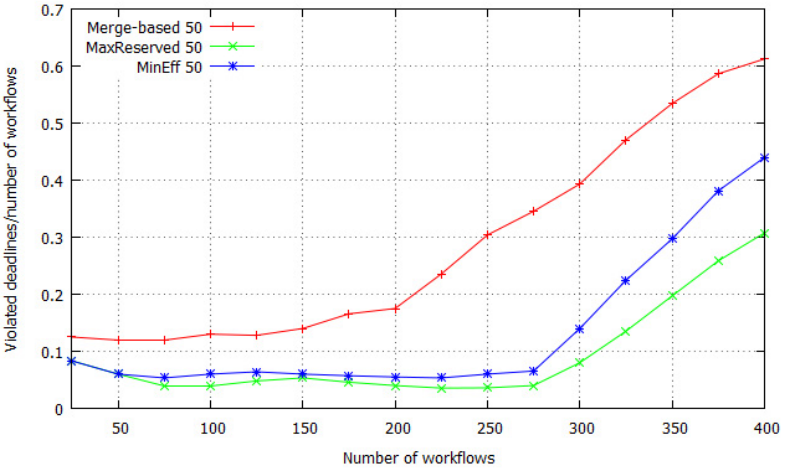

Figure 5. Ratio of violated deadlines to number of workflows for a tight makespan

Finally, Figure 4 shows the comparison of algorithms runtimes. Our experiments confirmed the estimates of computational complexity given in Section 4. Maximum execution times are:

for 400 5-tasks workflows: Merge-Based – 0.272 sec; average stage-based – 16,33 sec;
for 400 20-tasks workflows: Merge-Based – 1.338 sec; average stage-based – 55,75 sec;
for 400 50-tasks workflows: Merge-Based – 5.148 sec; average stage-based – 223,56 sec.

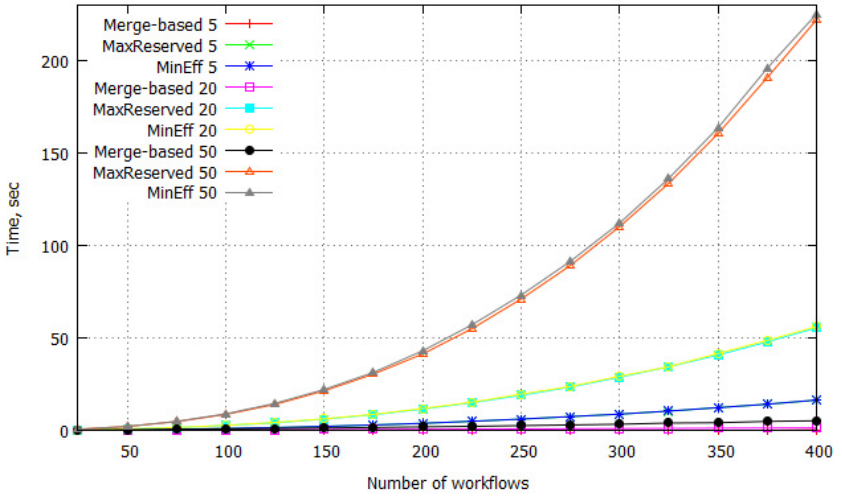

Figure 6. Execution times of algorithms

7 Conclusions and future work

In this paper we introduced an algorithm for a single deadline-constrained workflow scheduling for heterogeneous platforms with time windows and a staged scheme for a multiple deadline-constrained workflow scheduling. We run a series of experiments based on synthetic test examples in order to analyze and compare three proposed scheduling strategies in accordance with four metrics.

Simulation results indicate that the staged-based approach shows better results than merge-based for all schedule quality criteria (average reserved time, efficiency and meeting the deadlines). However, when computing environment is relatively free, we can use a merged-based algorithm due to its linear computational complexity and close to stage-based results.

Future work in this area might include the improvement of proposed scheduling formulation and algorithms considering the account costs of data transfer between the resource types and presence of data-parallel tasks. We aim to work out a fast rescheduling algorithm which should be applied in case of delay and tasks' failures, as well as to study the influence of the chosen prioritization criteria on the quality of obtained schedules. Also, the authors plan to embed the developed algorithms in a scheduling system of CLAVIRE [25] e-Science infrastructure platform.

This work was financially supported by Government of Russian Federation, Grant 074-U01.

References

[1] S. V. Kovalchuk, P. A. Smirnov, S. V. Maryin, T. N. Tchurov, and V. A. Karbovskiy, "Deadline-driven Resource Management within Urgent Computing Cyberinfrastructure," *Procedia Comput. Sci.*, vol. 18, pp. 2203–2212, Jan. 2013.

[2] A. V. Boukhanovsky and S. V. Ivanov, “Urgent Computing for Operational Storm Surge Forecasting in Saint-Petersburg,” *Procedia Comput. Sci.*, vol. 9, pp. 1704–1712, Jan. 2012.
[3] V.V. Krzhizhanovskaya, N.B. Melnikova, A.M. Chirkin, S.V. Ivanov, A.V. Boukhanovsky, P.M.A. Sloot. "Distributed simulation of city inundation by coupled surface and subsurface porous flow for urban flood decision support system". *Procedia Computer Science*, V. 18, pp. 1046-1056, 2013. http://dx.doi.org/10.1016/j.procs.2013.05.270
[4] S. V. Ivanov, S. S. Kosukhin, A. V. Kaluzhnaya, and A. V. Boukhanovsky, “Simulation-based collaborative decision support for surge floods prevention in St. Petersburg,” *J. Comput. Sci.*, vol. 3, no. 6, pp. 450–455, Nov. 2012.
[5] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, “Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads,” *2010 IEEE 3rd Int. Conf. Cloud Comput.*, pp. 228–235, Jul. 2010.
[6] T. A. L. Genez, L. F. Bittencourt, and E. R. M. Madeira, “Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels,” *2012 IEEE Network Operations and Management Symposium*. Ieee, pp. 906–912, 2012.
[7] H. Moens, K. Handekyn, and F. De Turck, “Cost-Aware Scheduling of Deadline-Constrained Task Workflows in Public Cloud Environments,” in *Integrated Network Management*, 2013, pp. 68–75.
[8] J. Y. J. Yu and R. Buyya, “A budget constrained scheduling of workflow applications on utility Grids using genetic algorithms,” *2006 Work. Work. Support Large-Scale Sci.*, pp. 1–10, 2006.
[9] W. Zheng and R. Sakellariou, “Budget-Deadline Constrained Workflow Planning for Admission Control,” *J. Grid Comput.*, vol. 11, no. 4, pp. 633–651, May 2013.
[10] H. Arabnejad and J. G. Barbosa, “List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.
[11] J. Yu, R. Buyya, and C. K. Tham, “Cost-based Scheduling of Scientific Workflow Applications on Utility Grids,” in *Proceedings of the First IEEE International Conference on e-Science and Grid Computing*, 2005, pp. 140–147.
[12] Y. Yuan, X. Li, Q. Wang, and Y. Zhang, “Bottom Level Based Heuristic for Workflow Scheduling in Grids,” *Chinese J. Comput.*, no. 31, pp. 282–290, 2008.
[13] A. Verma and S. Kaushal, “Deadline and Budget Distribution based Cost- Time Optimization Workflow Scheduling Algorithm for Cloud,” in *International Conference on Recent Advances and Future Trends in Information Technology2*, 2012, pp. 1–4.
[14] Q. Wang, X. Zhu, X. Li, and Y. Yuan, “Deadline division-based heuristic for cost optimization in workflow scheduling,” *Inf. Sci. (Ny).*, vol. 179, no. 15, pp. 2562–2575, Jul. 2009.
[15] M. Naghibzadeh and S. Abrishami, “Deadline-constrained workflow scheduling in software as a service Cloud,” *Sci. Iran.*, vol. 19, no. 3, pp. 680–689, Jun. 2012.
[16] D. H. J. Epema, M. Naghibzadeh, and S. Abrishami, “Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, Jan. 2013.
[17] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J. L. González-García, T. Röblitz, and J. M. Ramírez-Alcaraz, “Multiple Workflow Scheduling Strategies with User Run Time Estimates on a Grid,” *J. Grid Comput.*, vol. 10, no. 2, pp. 325–346, Mar. 2012.
[18] L. Zhu, Z. Sun, W. Guo, Y. Jin, W. Sun, and W. Hu, “Dynamic multi DAG scheduling algorithm for optical grid environment,” in *Network Architect*, 2007, vol. 6784, p. 67841F–67841F–11.
[19] H. Zhao and R. Sakellariou, “Scheduling Multiple DAGs onto Heterogeneous Systems,” in *Parallel and Distributing Processing Simposium, IPDPS’06*, 2006, p. 14.
[20] M. Mao and M. Humphrey, “Auto-scaling to minimize cost and meet application deadlines in cloud workflows,” *Proc. 2011 Int. Conf. High Perform. Comput. Networking, Storage Anal. - SC ’11*, p. 1, 2011.

[21] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," *2012 Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 1–11, Nov. 2012.

[22] J. Luo, F. Dong, and J. Zhang, "Scheduling of Scientific Workflow in Non-dedicated Heterogeneous Multicluster Platform," *J. Syst. Softw.*, vol. 86, no. 7, pp. 1806–1818, Jul. 2012.

[23] T. N'Takpe, F. Suter, and H. Casanova, "A Comparison of Scheduling Approaches for Mixed-Parallel Applications on Heterogeneous Platforms," in *Sixth International Symposium on Parallel and Distributed Computing (ISPDC'07)*, 2007, pp. 35–35.

[24] S. V Kovalchuk, P. A. Smirnov, K. V Knyazkov, A. S. Zagarskikh, and A. V Boukhanovsky, "Knowledge-based Expressive Technologies within Cloud Computing Environments," in *8th International Conference on Intelligent Systems and Knowledge Engineering (ISKE2013)*, 2013, pp. 1–10.

[25] K. V. Knyazkov, S. V. Kovalchuk, T. N. Tchurov, S. V. Maryin, and A. V. Boukhanovsky, "CLAVIRE: e-Science infrastructure for data-driven computing," *J. Comput. Sci.*, vol. 3, no. 6, pp. 504–510, Nov. 2012.